Monty Matlab Project Documentation

Jiangnan Huang, Zhiyuan Wu, Zhouyi Gu, Guang Chai
Department of Electrical and Computer Engineering, Chair of Data Processing
Technical University of Munich

I. Introduction

Human Activity Recognition (HAR)[1] is an active research field in pervasive computing. A HAR system has the main goal of analyzing human activities by observing and interpreting ongoing events successfully. Action recognition and prediction algorithms empower many real-world applications, such as visual surveillance, human-robot interaction, entertainment, and autonomous driving car. However, it's a challenging task to deal with a large number of time series data and find a solution to match data to recorded movements. In recent years, there specifically has been an increasing interest in recurrent neural network (RNN)[2] based approaches in the field of deep learning for time series data processing, classification or prediction in a fairly major way. In this project we propose a long short-term memory(LSTM)[3] network based on raw data collected from mobile phone accelerometer to classify walking gesture into two classes, the silly walk and the normal walk. The normal walk denotes the ordinary walking posture while the silly walk indicates any random weird movement, e.g. Figure 1. Both kinds of walk will be performed by our group and the data will be processed before used for model training and test. Specifications of data processing and model training can be found in second and third sections of this documentation respectively. We build a graphical user interface (GUI) for convenience in the end. We work on this project with Matlab Mobile App or Matlab Desktop and upload all data, script and program to Gitlab Group 4 Repository at https://gitlab.ldv.ei.tum.de/monty/g4-21.git.

II. Data Generation

The data generation is divided into two steps, data collection and selection and data extraction, and completed by all group members.

A. Data Collection and Selection

The raw data is collected in the following manner. We activate accelerometer sensor on the Matlab Mobile App with 60 Hz sampling rate and put the smartphone in back pocket. Then we start normal walk or silly walk for some time, e.g. 50 s, and deactivate the sensor. The acceleration data is uploaded to Matlab Cloud automatically. We repeat this procedure and each of us collects one piece of normal walk and three pieces of silly walk. The raw data is then selected by removing some temporal data in the beginning and in the end, when we use our smartphones instead of walking. For example, the first and the last 10 s are abolished to enhance the model performance and robustness later. The raw data includes acceleration in three directions and is saved in a 3xM matrix. Additionally for the next step, we add a 1xM true time duration array from 0s for each data point, which can be generated by product of sampling rate and amount of raw data points, and function like linespace(). Both data and time will be saved in file Group⟨k⟩_Walk⟨l⟩_N/S.mat, e.g. Group4_Walk1_S.mat for silly walk.

Fig. 1. Silly Walk Example[4]

B. Data Extraction

Before we use the data as input, we use a Matlab function extractData.m, which can be found in the root folder of repository, to create data slices according to instructions. This function accepts 4 parameters as input and returns a cell variable *X* for data slices and a categorical variable *Y* for label. The first input parameter *matFileContent* can be anything since it will be overwritten anyway. The second *filename* must be a string and the path and name must be correct. This function will include root folder and TrainingData, TestData folder and perform correctness check before slicing. Both terminal message and log file log_of_data_extraction.txt can be helpful when error occurs. If everything is proper, the new time array will be generated following resampling rate *samplingRateHz* and the data will be interpolated using interp1() with default setting. We use floor() to calculate rounded *windowWidth* and integer *window_amount*. With a *for* loop, the data is sliced into each window, specified by *windowWidthSeconds*, shifted 50% and saved in *X*. The label array *Y* is detected by name of file with help of contains() and generated using categorical() and ones(). This function can be tested by SillyWalkDetectionTest.p. For example, run this in terminal

```
test=SillyWalkDetectionTest
test.run()
```

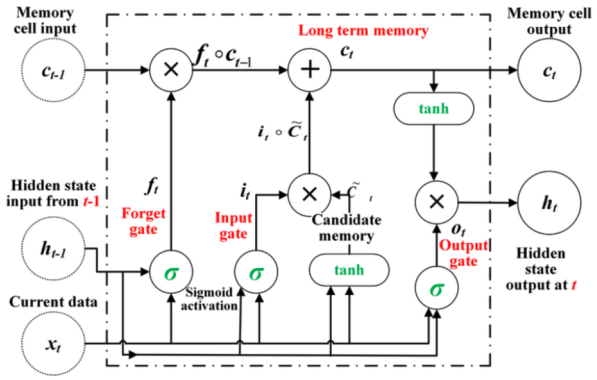

Fig. 2. LSTM architecture

III. Model Training and Classification

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate,e.g. Figure 2. They retain important data from the previous inputs and use that information to modify the current output, which makes them well suited to model temporal dynamics in activity time-series and deal with the vanishing gradient problem in numerous applications. For generating training samples 3x170 double training sample matrix *XTrain* and its corresponding class label categorical array *YTrain* are generated by specifying the variable *samplingRateHz* as 50 and *windowWidthSeconds* as 3.4 of the function extractData(). 20% silly or normal data will be used for test and the rest will be used for training. The training data is fed to a LSTM network, which comprises an input layer for sequences, a LSTM layer where the output mode is set to 'last' for sequence-to-label classification, followed by an dropout layer for overfitting reduction, a ReLU layer as activation function for non-linearity, a fully connected layer and a softmax layer for output normalization to probability distribution and finally a classification layer. The network structure is saved in variable *layers*. Since the dimension of each training sample is 3 and there are 2 class labels, the input size of the *sequenceInputLayer* is specified as 3 and number of classes of the *fullyConnectedLayer* as 2. With specifying *Shuffle* as 'once' in *trainingOptions* training data are shuffled once before training for smoother training accuracy curve. Besides, with 100 hidden units the LSTM network is trained with 32 mini batch for only 15 epochs to prevent overfitting and the model is stored in the file Model.mat. With the command

```
test = SillyWalkDetectionTest();
test.LoadModel=true;
test.run();
```

the model is loaded from mat file by unit test. The function classifyWalk() receives model and test samples *XTest*. With help of predict() function the probability of Normal walk for test samples in each cell array is generated. If the probability is more than 0.5, the test samples in corresponding cell array are labeled as Normal walk. After iterating all the cell arrays of *XTest*, the function classifyWalk() returns the categorical cell array *YPred* of predicted class labels for each sample in *XTest*. After finishing the whole unit test process, the reported accuracy/balanced accuracy can achieve around 95%. However the test accuracy on raw sample data is only around 75%, which means the test accuracy is strongly influenced by data generation, namely who recorded the data.

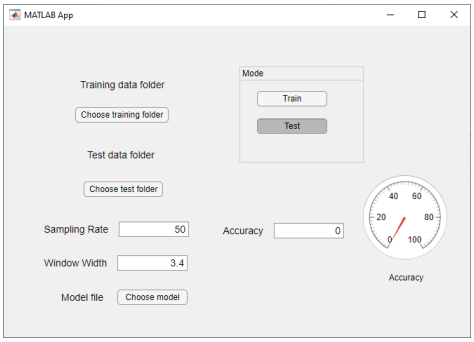

Fig. 3. GUI

IV. Graphical User Interface

As an alternative to calling functions in command line of Matlab, as shown in Fig. 3. we developed an graphical user interface(GUI) for choosing data folder, specifying the value of *samplingRateHz* and *windowWidthSeconds*, calling training and test function and displaying test accuracy. More specific, user can specify training data and test data folder in an pop-up menu by clicking corresponding button. If user do not specify any folder, the training and test data folder are defined as "TrainingData" and "TestData" in current folder by default respectively. Besides user can modify the default value of *samplingRateHz* and *windowWidthSeconds* with any positive number. After clicking the "Train" button the training process is displayed in another pop-up menu and model named "Model.mat" is automatically generated. With clicking "Test" button the test accuracy will be displayed in short time. With help of model choosing button user is able to test other model named "Model.mat" on test set as well.

V. Conclusion

The test accuracy shows that the LSTM network has unstable performance in simple Normal walk/Silly walk classification task and strongly affected by data generation. For better model performance more data collection and more dimension of data such as angular velocity are necessary.

References

[1] O. Chin Ann and B. Lau, "Human activity recognition: A review," pp. 389–393, 03 2015.
[2] R. Chalapathy, E. Z. Borzeshi, and M. Piccardi, "An investigation of recurrent neural architectures for drug name recognition," 2016.
[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.
[4] I. E. GmbH, "Monty matlab silly walks and matlab," June 2021.